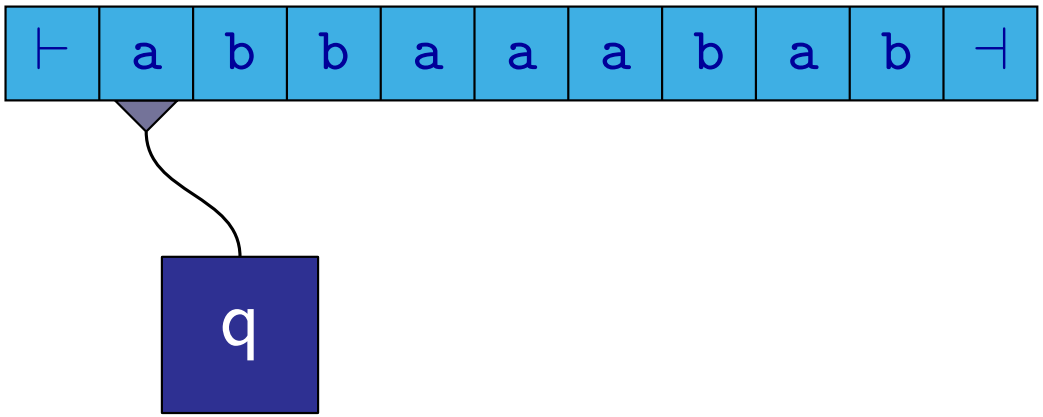
Lineárně omezený automat



Lineárně omezený automat:

- Turingův stroj, který využívá jen úsek pásky, na kterém je zapsáno vstupní slovo.
- Kolem slova jsou umístěny zarážky, které nemohou být přepsány.
- Z levé zarážky je možný pohyb jen vpravo, z pravé zarážky jen vlevo.

Lineárně omezený automat

- Lineárně omezené automaty můžeme uvažovat jak v **deterministické** tak v **nedeterministické** verzi.
- Standardně (tj. pokud není uvedeno jinak) se lineárně omezené automaty berou v nedeterministické verzi.
- Otázka, zda nedeterministické lineárně omezené automaty rozpoznávají stejnou nebo větší třídu jazyků než deterministické, je otevřená.

Generativní gramatika

Definice

Generativní gramatika je čtveřice $G = (\Pi, \Sigma, S, P)$, kde

- Π je konečná (neprázdná) množina neterminálů
- Σ je konečná (neprázdná) množina terminálů ($\Pi \cap \Sigma = \emptyset$)
- $S \in \Pi$ je počáteční neterminál
- P je konečná množina pravidel typu $\alpha \to \beta$, kde $\alpha \in (\Pi \cup \Sigma)^* \Pi (\Pi \cup \Sigma)^*$ a $\beta \in (\Pi \cup \Sigma)^*$.

- $\mu_1 \alpha \mu_2 \Rightarrow_G \mu_1 \beta \mu_2$ pokud $\alpha \to \beta$ je pravidlo v P
- $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
AB &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
AB &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

S

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$\begin{aligned} \underline{S} &\rightarrow aABbBB \\ AB &\rightarrow BA \\ bBB &\rightarrow aABA \\ BAb &\rightarrow CbA \\ BAb &\rightarrow BBbCA \\ aAb &\rightarrow baa \\ BB &\rightarrow b \\ CA &\rightarrow abB \\ Cb &\rightarrow aa \end{aligned}$$

$\underline{S}$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$\begin{aligned} \underline{S} &\rightarrow \underline{aABbBB} \\ AB &\rightarrow BA \\ bBB &\rightarrow aABA \\ BAb &\rightarrow CbA \\ BAb &\rightarrow BBbCA \\ aAb &\rightarrow baa \\ BB &\rightarrow b \\ CA &\rightarrow abB \\ Cb &\rightarrow aa \end{aligned}$$

$\underline{S} \Rightarrow \underline{aABbBB}$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
AB &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

$S \Rightarrow aABbBB$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
\underline{AB} &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

$S \Rightarrow a\underline{AB}bBB$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
\underline{AB} &\to \underline{BA} \\
bBB &\to aABA \\
BAb &\to CbA \\
BAb &\to BBbCA \\
aAb &\to baa \\
BB &\to b \\
CA &\to abB \\
Cb &\to aa
\end{aligned}
$$

$S \Rightarrow a\underline{AB}bBB \Rightarrow a\underline{BA}bBB$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
AB &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAbBB$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
AB &\to BA \\
bBB &\to aABA \\
BAb &\to CbA \\
BAb &\to BBbCA \\
aAb &\to baa \\
\underline{BB} &\to b \\
CA &\to abB \\
Cb &\to aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAb\underline{BB}$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
AB &\to BA \\
bBB &\to aABA \\
BAb &\to CbA \\
BAb &\to BBbCA \\
aAb &\to baa \\
\underline{BB} &\to \underline{b} \\
CA &\to abB \\
Cb &\to aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAb\underline{BB} \Rightarrow aBAb\underline{b}$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
AB &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow aBAbb$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
AB &\to BA \\
bBB &\to aABA \\
\underline{BAb} &\to CbA \\
BAb &\to BBbCA \\
aAb &\to baa \\
BB &\to b \\
CA &\to abB \\
Cb &\to aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow a\underline{BAb}b$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
AB &\to BA \\
bBB &\to aABA \\
\underline{BAb} &\to \underline{CbA} \\
BAb &\to BBbCA \\
aAb &\to baa \\
BB &\to b \\
CA &\to abB \\
Cb &\to aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow a\underline{BAb}b \Rightarrow a\underline{CbA}b$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$

$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
AB &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow aBAbb \Rightarrow aCbAb$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
AB &\to BA \\
bBB &\to aABA \\
BAb &\to CbA \\
BAb &\to BBbCA \\
aAb &\to baa \\
BB &\to b \\
CA &\to abB \\
\underline{Cb} &\to aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow aBAbb \Rightarrow a\underline{Cb}Ab$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
AB &\to BA \\
bBB &\to aABA \\
BAb &\to CbA \\
BAb &\to BBbCA \\
aAb &\to baa \\
BB &\to b \\
CA &\to abB \\
\underline{Cb} &\to \underline{aa}
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow aBAbb \Rightarrow a\underline{Cb}Ab \Rightarrow a\underline{aa}Ab$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
AB &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

$$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow aBAbb \Rightarrow aCbAb \Rightarrow aaaAb$$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
AB &\to BA \\
bBB &\to aABA \\
BAb &\to CbA \\
BAb &\to BBbCA \\
\underline{aAb} &\to baa \\
BB &\to b \\
CA &\to abB \\
Cb &\to aa
\end{aligned}
$$

$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow aBAbb \Rightarrow aCbAb \Rightarrow aa\underline{aAb}$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\to aABbBB \\
AB &\to BA \\
bBB &\to aABA \\
BAb &\to CbA \\
BAb &\to BBbCA \\
\underline{aAb} &\to \underline{baa} \\
BB &\to b \\
CA &\to abB \\
Cb &\to aa
\end{aligned}
$$

$$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow aBAbb \Rightarrow aCbAb \Rightarrow aa\underline{aAb} \Rightarrow aa\underline{baa}$$

Generativní gramatika

Příklad: Generativní gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABbBB \\
AB &\rightarrow BA \\
bBB &\rightarrow aABA \\
BAb &\rightarrow CbA \\
BAb &\rightarrow BBbCA \\
aAb &\rightarrow baa \\
BB &\rightarrow b \\
CA &\rightarrow abB \\
Cb &\rightarrow aa
\end{aligned}
$$

$$S \Rightarrow aABbBB \Rightarrow aBAbBB \Rightarrow aBAbb \Rightarrow aCbAb \Rightarrow aaaAb \Rightarrow aabaa$$

Chomského hierarchie

Podle tvaru pravidel, která v gramatice povolíme, dělíme gramatiky na gramatiky typu:

- 0 – Obecné **generativní gramatiky** – pravidla bez omezení
- 1 – **Kontextové gramatiky** – pravidla tvaru $\alpha X \beta \to \alpha \gamma \beta$, kde $|\gamma| \geq 1$ (Výjimka $S \to \varepsilon$, ale S pak není na pravé straně žádného pravidla.)
- 2 – **Bezkontextové gramatiky** – pravidla tvaru $X \to \gamma$
- 3 – **Regulární gramatiky** – pravidla tvaru $X \to wY$ nebo $X \to w$

$$\alpha, \beta, \gamma \in (\Pi \cup \Sigma)^*, X \in \Pi, w \in \Sigma^*$$

Kontextová gramatika

Příklad: Kontextová gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow aABb \\
AB &\rightarrow ABb \\
aBA &\rightarrow aAAA \\
AAa &\rightarrow Aaba \\
bA &\rightarrow bBBa \\
abA &\rightarrow abab \\
A &\rightarrow BA
\end{aligned}
$$

Kontextová gramatika

Příklad: Kontextová gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
\underline{S} &\longrightarrow \underline{aABb} \\
A\underline{B} &\longrightarrow A\underline{Bb} \\
a\underline{B}A &\longrightarrow a\underline{AA}A \\
A\underline{A}a &\longrightarrow A\underline{ab}a \\
b\underline{A} &\longrightarrow b\underline{BBa} \\
ab\underline{A} &\longrightarrow ab\underline{ab} \\
\underline{A} &\longrightarrow \underline{BA}
\end{aligned}
$$

Regulární gramatika

Příklad: Regulární gramatika $G = (\Pi, \Sigma, S, P)$

$\Pi = \{S, A, B, C\}$
$\Sigma = \{a, b\}$

Množina pravidel P:

$$
\begin{aligned}
S &\rightarrow abA \mid bB \mid b \mid \varepsilon \\
A &\rightarrow bC \mid abB \\
B &\rightarrow aB \mid aC \mid bbC \\
C &\rightarrow aA \mid bC \mid ba
\end{aligned}
$$

Chomského hierarchie

- Jazyk je typu i, jestliže jej generuje nějaká gramatika typu i.

Pojmenování jednotlivých typů jazyků:

- Typ 0 – Rekurzivně spočetné
- Typ 1 – Kontextové
- Typ 2 – Bezkontextové
- Typ 3 – Regulární

Chomského hierarchie

Jednotlivé typy jazyků přesně odpovídají třídám jazyků rozpoznávaným následujícími typy automatů:

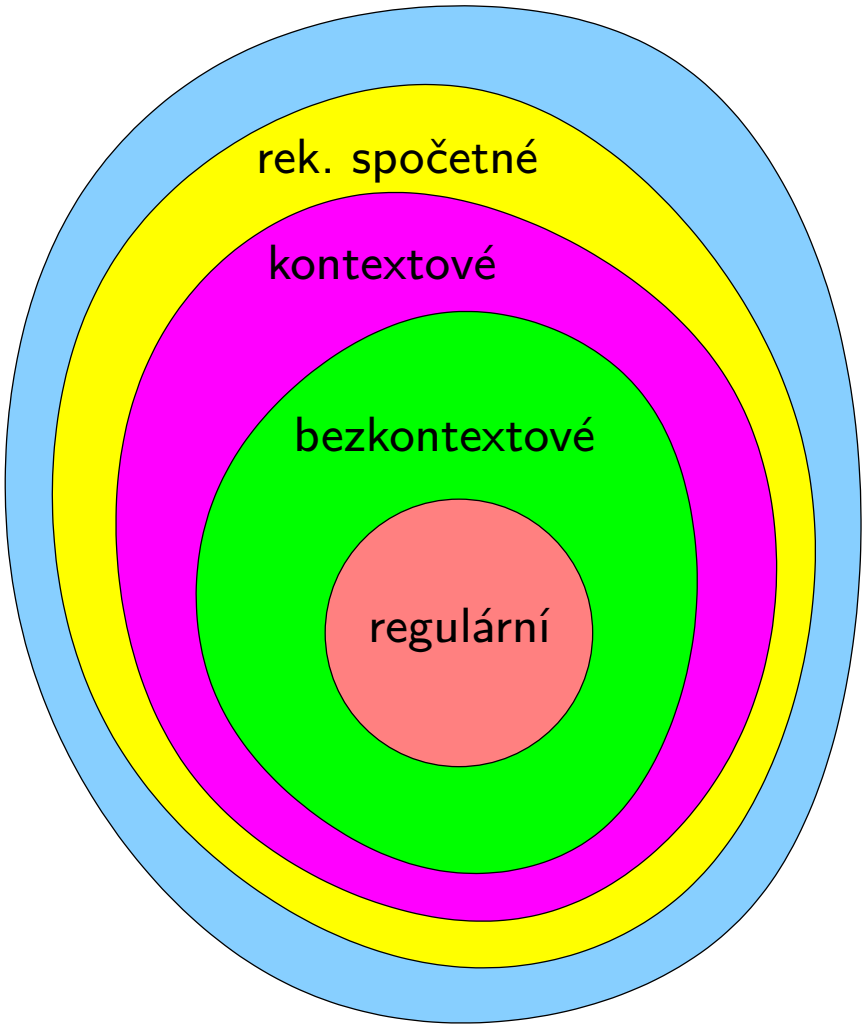
- Typ 0 – Turingovy stroje (deterministické nebo nedeterministické)
- Typ 1 – Lineárně omezené automaty (nedeterministické)
- Typ 2 – Zásobníkové automaty (nedeterministické)
- Typ 3 – Konečné automaty (deterministické nebo nedeterministické)

Chomského hierarchie

Vztahy mezi jednotlivými typy jazyků, gramatik a automatů jsou shrnuty v následující tabulce:

Typ	**Jazyky**	**Gramatiky**	**Automaty**
0	rekurzivně spočetné	obecné	Turingovy stroje
1	kontextové	kontextové	lin. omezené automaty
2	bezkontextové	bezkontextové	zásobníkové automaty
3	regulární	regulární	konečné automaty

Chomského hierarchie



- Příklad regulárního jazyka:
  $L_1 = \{a^m b^n \mid m \geq 0,\ n \geq 0\}$
- Příklad bezkontextového jazyka, který není regulární:
  $L_2 = \{a^n b^n \mid n \geq 0\}$
- Příklad kontextového jazyka, který není bezkontextový:
  $L_3 = \{a^n b^n c^n \mid n \geq 0\}$

Vyčíslitelnost a složitost

Co je to algoritmus?

Algoritmus

Algoritmus je mechanický postup skládající se z nějakých jednoduchých elementárních kroků, který pro nějaký zadaný **vstup** vyprodukuje nějaký **výstup**.

Algoritmus může být zadán:

- slovním popisem v přirozeném jazyce
- pseudokódem
- jako počítačový program v nějakém programovacím jazyce
- jako hardwarový obvod
- ...

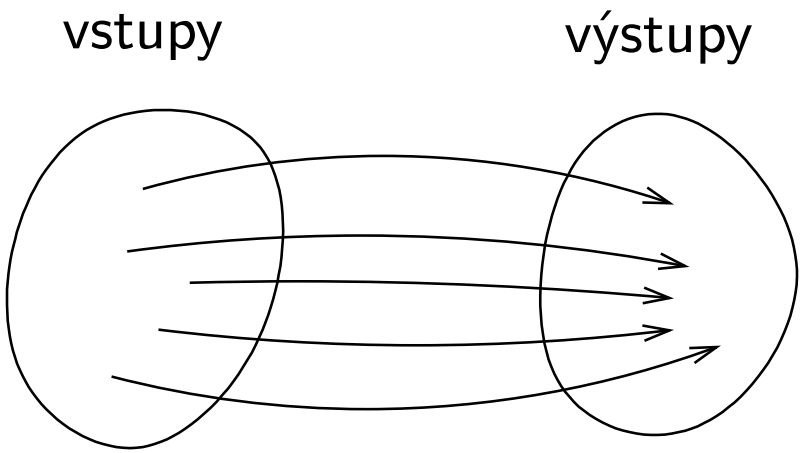
Algoritmy slouží k řešení různých **problémů**.

Problémy

Problém

V zadání **problému** musí být určeno:

- co je množinou možných vstupů
- co je množinou možných výstupů
- jaký je vztah mezi vstupy a výstupy



Příklady problémů

Problém „Třídění“

Vstup: Sekvence prvků $a_1, a_2, \ldots, a_n$.

Výstup: Prvky sekvence $a_1, a_2, \ldots, a_n$ seřazené od nejmenšího po největší.

Příklad:

- Vstup: $8, 13, 3, 10, 1, 4$
- Výstup: $1, 3, 4, 8, 10, 13$

Poznámka: Konkrétní vstup nějakého problému se nazývá **instance** problému.

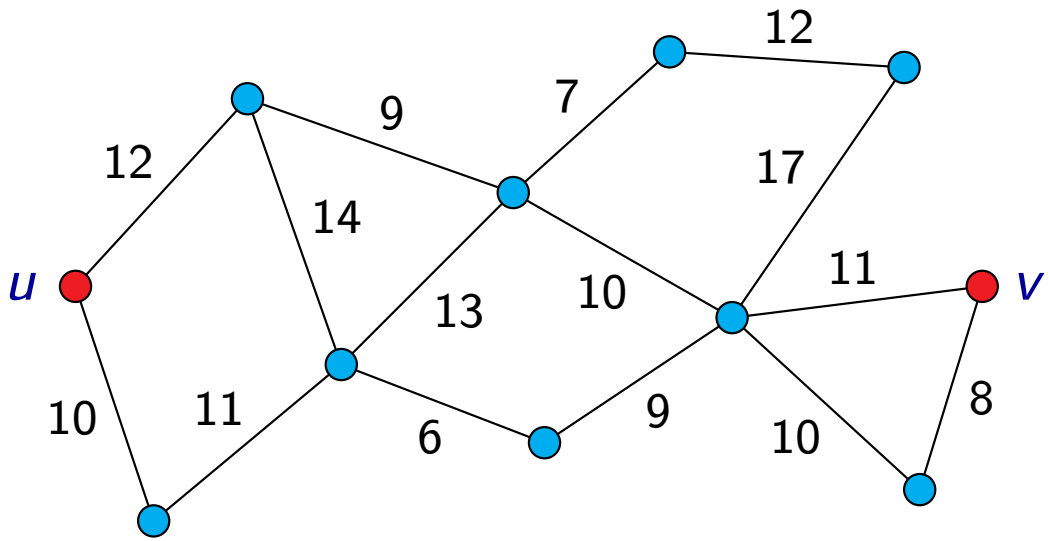
Příklady problémů

Problém „Hledání nejkratší cesty v (neorientovaném) grafu“

Vstup: Neorientovaný graf $G = (V, E)$ s ohodnocením hran, a dvojice vrcholů $u, v \in V$.

Výstup: Nejkratší cesta z vrcholu u do vrcholu v.

Příklad:



Další příklady problémů

Problém „Prvočíselnost“

Vstup: Přirozené číslo n.

Výstup: ANO pokud je n prvočíslo, NE v opačném případě.

Poznámka: Přirozené číslo n je **prvočíslo**, pokud je větší než 1 a je dělitelné beze zbytku pouze čísly 1 a n.

Prvních několik prvočísel: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, ...

Rozhodovací problémy

Situace, kdy množina výstupů je $\{\text{Ano}, \text{Ne}\}$ je poměrně častá. Takovým problémům se říká **rozhodovací problémy**.

Rozhodovací problémy většinou specifikujeme tak, že místo popisu toho, co je výstupem, uvedeme otázku.

Příklad:

Problém „Prvočíselnost“

Vstup: Přirozené číslo n.

Otázka: Je n prvočíslo?

Optimalizační problémy

Dalším speciálním případem jsou tzv. optimalizační problémy.

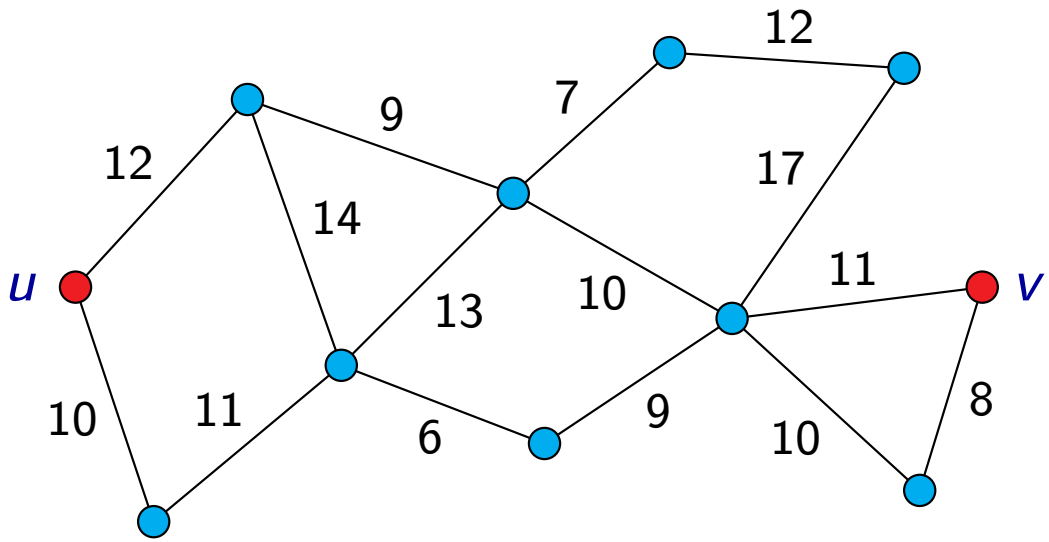
Optimalizační problém je problém, kde je úkolem vybrat z nějaké množiny přípustných řešení takové řešení, které je v nějakém ohledu optimální.

Optimalizační problémy

Dalším speciálním případem jsou tzv. optimalizační problémy.

Optimalizační problém je problém, kde je úkolem vybrat z nějaké množiny přípustných řešení takové řešení, které je v nějakém ohledu optimální.

Příklad: V problému „Hledání nejkratší cesty v grafu“ je množina všech přípustných řešení tvořena všemi cestami z vrcholu u do vrcholu v. Kritériem, podle kterého cesty hodnotíme, je délka cesty.



Řešení problémů

Řešení problému

Algoritmus **korektně řeší** daný problém, když:

1. Se pro libovolný vstup daného problému (libovolnou vstupní instanci) po konečném počtu kroků zastaví.
2. Vyprodukuje výstup z množiny možných výstupů, který vyhovuje podmínkám uvedeným v zadání problému.

Pro jeden problém může existovat celá řada algoritmů, které jej korektně řeší.

Poznámka: Množina vstupních instancí bývá typicky nekonečná.

Kódování vstupu a výstupu

Většinou předpokládáme, že vstupy i výstupy jsou kódovány jako slova v nějaké abecedě Σ.

Příklad: Například u problému „Třídění“ bychom mohli zvolit jako abecedu $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ,\}$.

Vstupem by pak mohlo být například slovo

$$826,13,3901,101,128,562$$

a výstupem slovo

$$13,101,128,562,826,3901$$

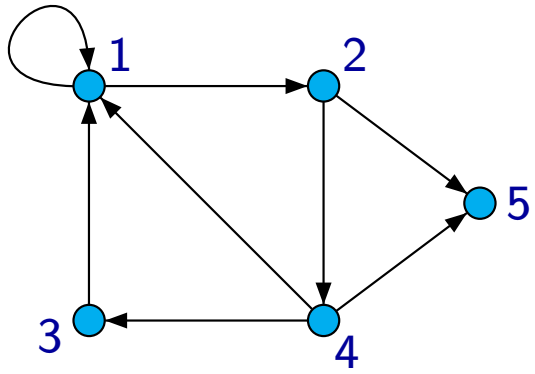
Kódování vstupu a výstupu

Poznámka: Ne každé slovo ze Σ^* musí reprezentovat nějaký vstup. Kódování bychom ale měli zvolit tak, abychom byli schopni snadno poznat ta slova, která nějaký vstup reprezentují.

Kódování vstupu a výstupu

Příklad: Pokud je vstupem nějakého problému například graf, můžeme ho reprezentovat jako seznam vrcholů a hran:

Například následující graf



můžeme reprezentovat jako slovo

$$(1,2,3,4,5),((1,2),(2,4),(4,3),(3,1),(1,1),(2,5),(4,5),(4,1))$$

v abecedě $\Sigma = \{0,1,2,3,4,5,6,7,8,9,,,(,)\}$.

Kódování vstupu a výstupu

Můžeme se omezit na případ, kdy jsou vstupy i výstupy kódovány jako slova v abecedě $\{0, 1\}$ (tj. jako sekvence bitů).

Symboly jakékoli jiné abecedy lze reprezentovat jako sekvence bitů.

Příklad: Abeceda $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}, \mathtt{g}\}$

$$
\begin{aligned}
\mathtt{a} &\leftrightarrow 001 \\
\mathtt{b} &\leftrightarrow 010 \\
\mathtt{c} &\leftrightarrow 011 \\
\mathtt{d} &\leftrightarrow 100 \\
\mathtt{e} &\leftrightarrow 101 \\
\mathtt{f} &\leftrightarrow 110 \\
\mathtt{g} &\leftrightarrow 111
\end{aligned}
$$

Slovo '$\mathtt{defb}$' můžeme reprezentovat jako '100101110010'.

Funkce realizovaná algoritmem

Můžeme tedy říct, že každý algoritmus realizuje výpočet hodnot nějaké funkce

$$f : \Sigma^* \to \Sigma^*$$

kde $\Sigma = \{0, 1\}$.

Alternativně se také na algoritmy můžeme dívat tak, že realizují výpočet nějaké funkce

$$f : \mathbb{N} \to \mathbb{N}$$

kde $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ je množina přirozených čísel, neboť na každou sekvenci bitů se můžeme dívat jako na zápis čísla v binární soustavě.

Funkce realizovaná algoritmem

Funkce

$$f : \Sigma^* \to \Sigma^* \qquad \text{resp.} \qquad f : \mathbb{N} \to \mathbb{N}$$

realizovaná nějakým algoritmem nemusí být nutně totální, může být částečná. Hodnota $f(x)$ není definovaná například v případě, že se daný algoritmus pro vstup x nikdy nezastaví.

Poznámka: Funkce $f : A \to B$ je **totální**, jestliže hodnota $f(x)$ je definovaná pro libovolné $x \in A$, a **částečná**, jestliže pro některá $x \in A$ nemusí být hodnota $f(x)$ definovaná.

Algoritmy a problémy

Stejně jako na algoritmy i na problémy se můžeme dívat jako na funkce typu:

$$f : \Sigma^* \to \Sigma^* \qquad \text{resp.} \qquad f : \mathbb{N} \to \mathbb{N}$$

Jestliže algoritmus realizuje nějakou funkci f', řekneme, že tento algoritmus **řeší** problém popsaný funkcí f, jestliže se pro libovolný vstup x zastaví s tím, že

$$f(x) = f'(x)$$

Rozhodovací problémy a jazyky

Na rozhodovací problémy můžeme pohlížet jako na jazyky.

Jazyk odpovídající danému rozhodovacímu problému je množina těch slov ze Σ^*, která reprezentují ty vstupy, pro něž je odpověď ANO.

Příklad: Jazyk L tvořený těmi slovy ze $\{0,1\}^*$, která jsou binárním zápisem nějakého prvočísla.

Například $101 \in L$, ale $110 \notin L$.

Rozhodovací problémy

Často se při zkoumání algoritmů a problému omezujeme jen na rozhodovací problémy.

Není to však na úkor obecnosti, neboť libovolný obecný problém je možné vhodným způsobem přeformulovat jako rozhodovací problém, s tím, že když najdeme algoritmus, který by řešil tento rozhodovací problém, tak bychom snadno sestrojili algoritmus, který by řešil původní problém, a naopak.

Rozhodovací problémy

Pokud například máme problém P, kde:

- vstupy jsou prvky z nějaké množiny X
- výstupy jsou slova z $\{0,1\}^*$

můžeme tento problém přeformulovat jako následující rozhodovací problém:

Problém

Vstup: Prvek $x \in X$ a číslo k.

Otázka: Když z je výstup, který odpovídá vstupu x problému P, má k-tý bit slova z hodnotu 1?

Algoritmicky řešitelné problémy

Předpokládejme, že máme dán nějaký problém P.

Jestliže existuje nějaký algoritmus, který řeší problém P, pak říkáme, že problém P je **algoritmicky řešitelný**.

Jestliže P je rozhodovací problém a jestliže existuje nějaký algoritmus, který problém P řeší, pak říkáme, že problém P je **rozhodnutelný**.

Když chceme ukázat, že problém P je algoritmicky řešitelný, stačí ukázat nějaký algoritmus, který ho řeší (a případně ukázat, že daný algoritmus problém P skutečně řeší).

Algoritmicky řešitelné problémy

U mnohých problémů je na první pohled zřejmé, že jsou algoritmicky řešitelné, jako třeba:

- Třídění
- Hledání nejkratší cesty v grafu
- Prvočíselnost

kde stačí probrat všechny možnosti (kterých je ve všech těchto případech konečně mnoho), i když takový triviální algoritmus založený na řešení **hrubou silou** nemusí být zrovna efektivní.

Na druhou stranu existuje celá řada problémů, u kterých to tak jasné není.

- Najít algoritmus a dokázat, že řeší daný problém, může být velmi netriviální úkol.
- Algoritmus, který by řešil daný problém, nemusí vůbec existovat.

Nutnost upřesnění pojmu „algoritmus“

Dosavadní definice pojmu algoritmus byla poněkud vágní.

Pokud bychom pro nějaký problém chtěli ukázat, že neexistuje algoritmus, který by daný problém řešil, tak by to s takovouto neurčitou definicí pojmu algoritmus asi nešlo.

Intuitivně chápeme, co by měl mít algoritmus za vlastnosti:

- Měl by se skládat z jednoduchých kroků, které je možno vykonávat „mechanicky“, bez porozumění problému.
- Objekty, se kterými algoritmus pracuje, i prováděné operace by měly být konečné.

Algoritmus realizovaný počítačem

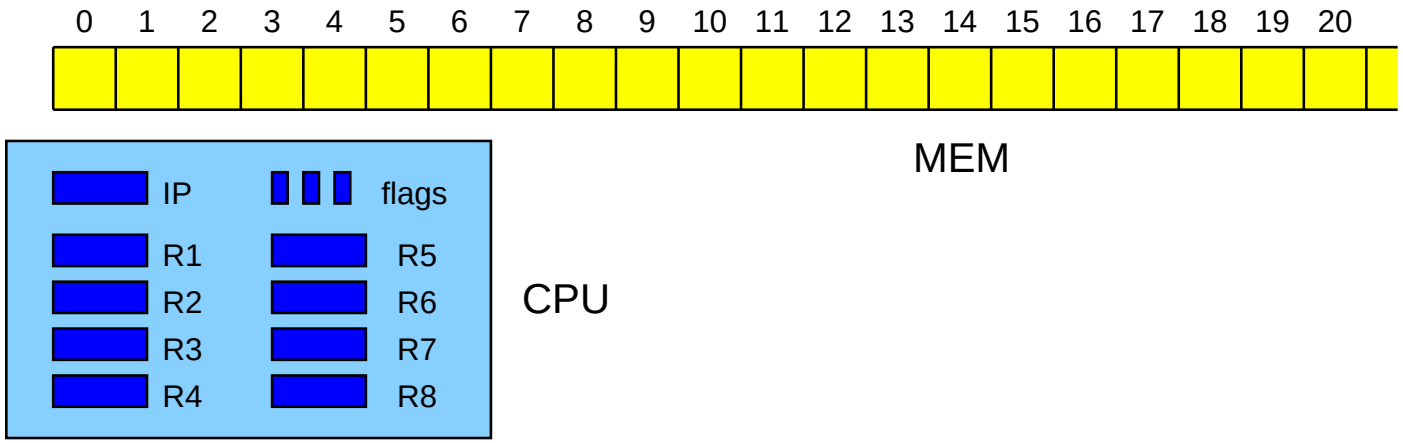
Například program v libovolném programovacím jazyce dané vlastnosti zcela jistě má.

Ať už je program napsán v jakémkoliv programovacím jazyce, jsou jeho instrukce nakonec prováděny na hardwaru nějakého konkrétního počítače na úrovni instrukcí procesoru.

Je tedy jasné, že **každý** program v **každém** programovacím jazyce bychom mohli zapsat jako program tvořený pouze instrukcemi strojového kódu nějakého procesoru.

Činnost počítače

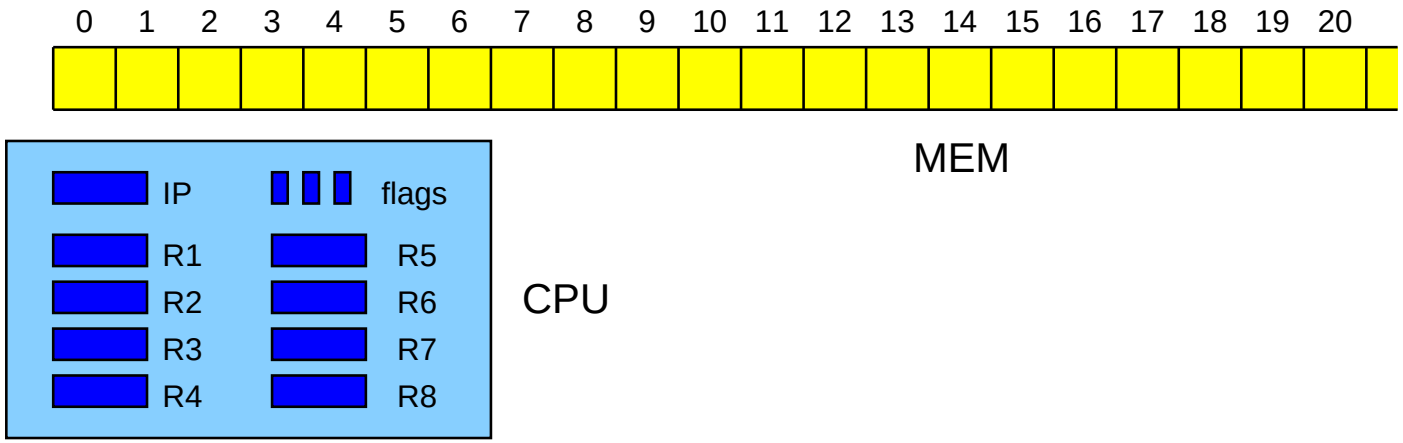
Typická architektura naprosté většiny počítačů vypadá následovně:



- Počítač má **paměť** skládající se z velkého množství paměťových buněk.
- Každá buňka může obsahovat číslo určité velikosti, typicky 1 byte (8 bitů), tj. číslo v rozsahu $0 \ldots 255$.
- Buňky jsou očíslovány. Číslo buňky se nazývá její **adresa**.

Činnost počítače

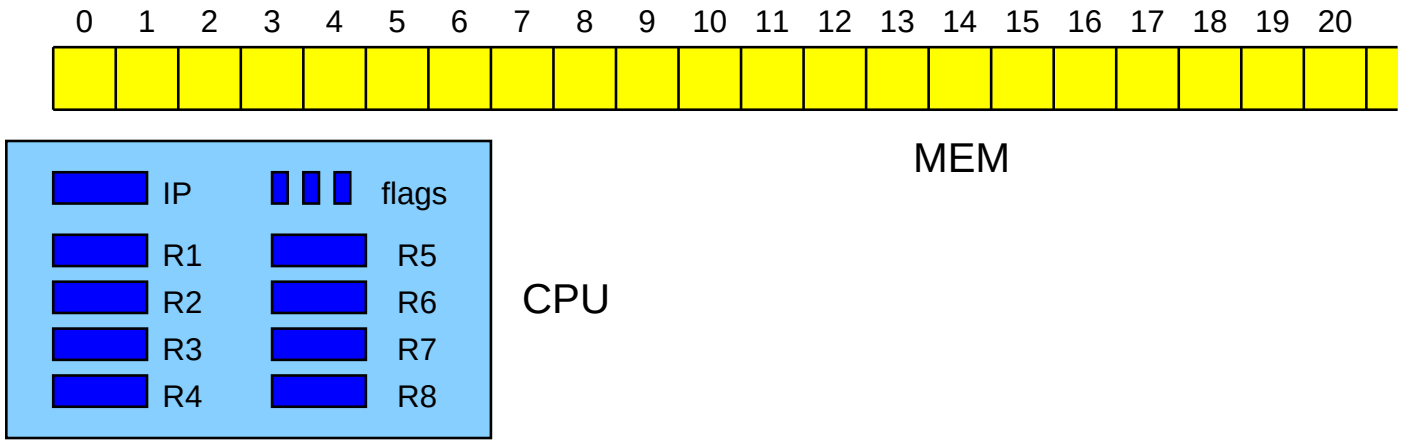
Typická architektura naprosté většiny počítačů vypadá následovně:



- Instrukce jsou uloženy v paměti (každá instrukce má svůj číselný **kód**) a jsou sekvenčně vykonávány **procesorem**.
- Procesor udržuje tzv. **čítač instrukcí** IP, který obsahuje adresu aktuálně prováděné instrukce.
- Procesor načte instrukci z adresy určené IP, zvětší IP o délku načtené instrukce a provede danou instrukci.

Činnost počítače

Typická architektura naprosté většiny počítačů vypadá následovně:



- Procesor obsahuje několik **registrů** pevné délky (např. 32 nebo 64 bitů).
- Většina operací je prováděna na registrech.
- Procesor obsahuje **příznaky (flags)**, které umožňují testovat výsledek poslední operace (např. přetečení, jestli je výsledek nula apod.).

Činnost počítače

Typické instrukce:

- Načtení obsahu paměťové buňky (resp. několika po sobě jdoucích buněk) do některého registru (LOAD).
- Uložení obsahu registru do některé paměťové buňky (resp. několika po sobě jdoucích buněk) (STORE).

  Poznámka: Adresa buňky je buď přímá (tj. je přímo součástí instrukce) nebo nepřímá (uložená v některém registru, případně spočítaná z obsahu jednoho nebo několika registrů).
- Načtení obsahu jednoho registru do jiného registru (MOV).
- Aritmetické instrukce (ADD, SUB, MUL, DIV, NEG, CMP, INC, DEC, ...).
- Logické instrukce (AND, OR, XOR, NOT, ...).
- Bitové posuny a rotace (SHL, SHR, ...)

Činnost počítače

Typické instrukce (pokračování):

- Nepodmíněný skok (JMP).

  Poznámka: Cílová adresa může být přímá nebo nepřímá.

- Podmíněné skoky (JZERO, JGTZ, ...).
- Volání podprogramů (CALL, RET).
- Různé speciální instrukce – práce se vstupem a výstupem, obsluha přerušení, mody činnosti procesoru, řízení přístupu do paměti, stránkování apod.

Činnost počítače

Příklad: Instrukce zapsaná ve vyšším programovacím jazyce jako

$$x = y + 2$$

může být realizována následující sekvencí instrukcí určitého (hypotetického) procesoru:

```
LOAD    R5,[0x001b7c44]
ADD     R5,2
STORE   [0x001b7c38],R5
```

Poznámka: Předpokládáme, že proměnná x je uložena na adrese 0x001b7c38 a proměnná y na adrese 0x001b7c44.

Stroj RAM

Stroj RAM (Random Access Machine) je idealizovaný model počítače.

Rozdíly oproti skutečnému počítači:

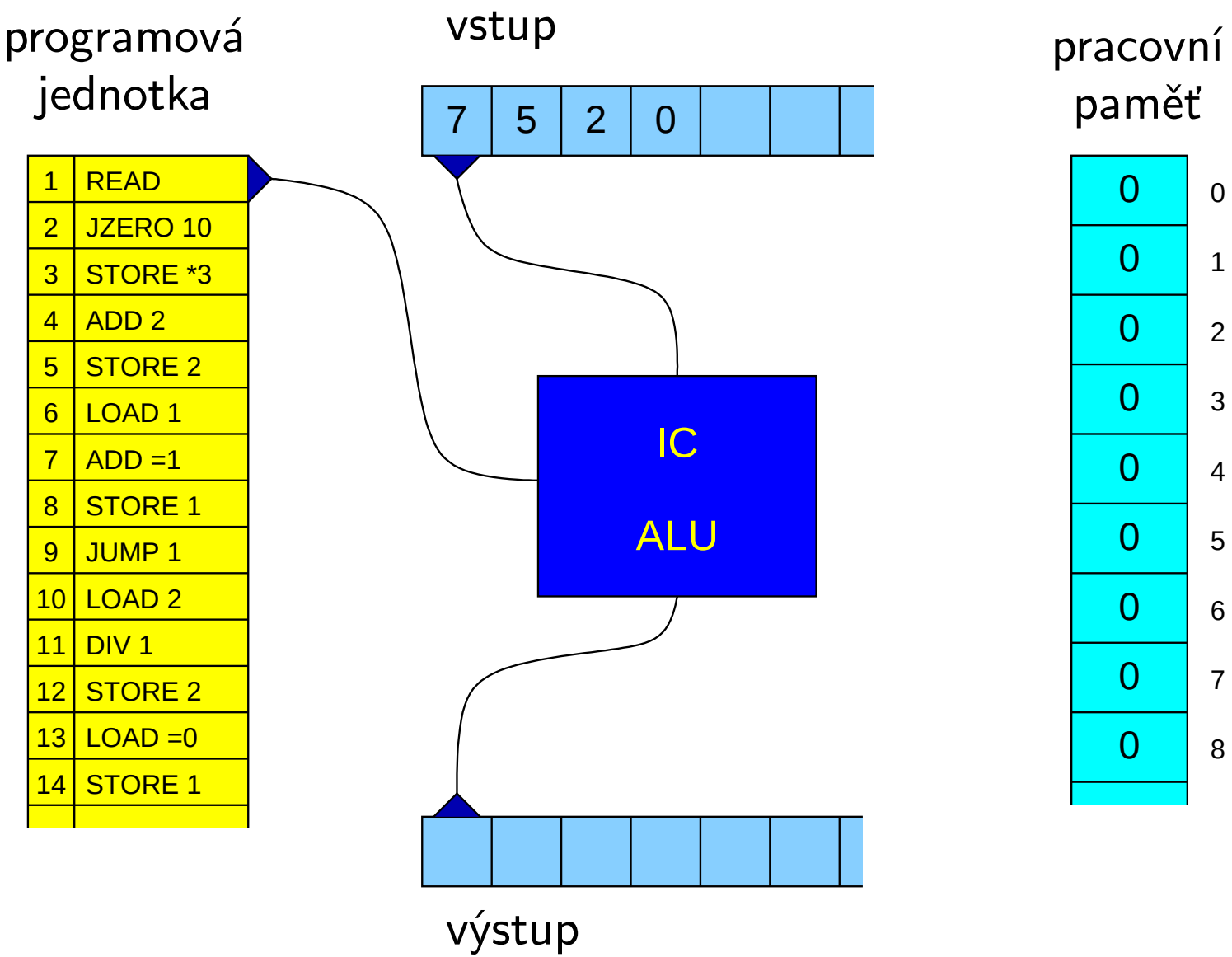
- Velikost paměti není omezena (adresa může být libovolné přirozené číslo).
- Velikost obsahu jednotlivých buněk není omezena (buňka může obsahovat libovolné celé číslo).
- Čte data sekvenčně ze vstupu, který je tvořen sekvencí celých čísel. Ze vstupu lze pouze číst.
- Zapisuje data sekvenčně na výstup, který je tvořen sekvencí celých čísel. Na výstup je možné pouze zapisovat.

Stroj RAM

RAM (Random Access Machine) neboli **počítač s libovolným přístupem** se skládá z těchto částí:

- **Programová jednotka** – obsahuje program stroje RAM a ukazatel na právě prováděnou instrukci
- **Pracovní paměť** tvořená buňkami očíslovanými $0, 1, 2, \ldots$; obsah buněk je možno číst i do nich zapisovat
- **Vstupní páska** – je z ní možné pouze číst
- **Výstupní páska** – je na ni možno pouze zapisovat

Stroj RAM


programová jednotka
vstup
pracovní paměť
1 READ
2 JZERO 10
3 STORE *3
4 ADD 2
5 STORE 2
6 LOAD 1
7 ADD =1
8 STORE 1
9 JUMP 1
10 LOAD 2
11 DIV 1
12 STORE 2
13 LOAD =0
14 STORE 1
7 5 2 0
IC
ALU
0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8
výstup

Stroj RAM

Buňky 0 a 1 mají speciální význam a slouží jako „registry“ stroje RAM:

- **Buňka 0** – **pracovní registr** (akumulátor) – registr, který je jedním z operandů většiny instrukcí a do kterého se ukládá výsledek většiny operací.
- **Buňka 1** – **indexový registr** – je použit při nepřímém adresování.

Tvary **operandů** instrukcí ($i \in \mathbb{N}$):

tvar	hodnota operandu
$=i$	přímo číslo udané zápisem i
i	číslo obsažené v buňce s adresou i
$*i$	číslo v buňce s adresou $i + j$, kde j je aktuální obsah indexového registru

Příklad:

`LOAD <op>`

načte obsah operandu `<op>` do pracovního registru (tj. do buňky číslo 0).

`LOAD =5`	–	uloží do pracovního registru hodnotu 5
`LOAD 5`	–	uloží do pracovního registru obsah buňky číslo 5
`LOAD *5`	–	uloží do pracovního registru obsah buňky číslo $5+j$, kde j je aktuální obsah indexového registru

Stroj RAM

Instrukce vstupu a výstupu (jsou bez operandu):

- `READ` – do pracovního registru se uloží číslo, které je v políčku snímaném vstupní hlavou, a vstupní hlava se posune o jedno políčko doprava
- `WRITE` – výstupní hlava zapíše do snímaného políčka výstupní pásky obsah pracovního registru a posune se o jedno políčko doprava

Instrukce přesunu v paměti:

- `LOAD <op>` – do pracovního registru se načte hodnota operandu
- `STORE <op>` – hodnota operandu se přepíše obsahem pracovního registru (zde se nepřipouští operand tvaru $=i$)

Stroj RAM

Instrukce aritmetických operací:

- `ADD <op>` – číslo v pracovním registru se zvýší o hodnotu operandu (tedy přičte se k němu hodnota operandu)
- `SUB <op>` – od čísla v pracovním registru se odečte hodnota operandu
- `MUL <op>` – číslo v pracovním registru se vynásobí hodnotou operandu
- `DIV <op>` – číslo v pracovním registru se celočíselně vydělí hodnotou operandu (do pracovního registru se uloží výsledek příslušného celočíselného dělení)

Stroj RAM

Instrukce skoku:

`JUMP <návěští>` – výpočet bude pokračovat instrukcí určenou návěštím

`JZERO <návěští>` – je-li obsahem pracovního registru číslo 0, bude výpočet pokračovat instrukcí určenou návěštím; v opačném případě bude pokračovat následující instrukcí

`JGTZ <návěští>` – je-li číslo v pracovním registru kladné, bude výpočet pokračovat instrukcí určenou návěštím; v opačném případě bude pokračovat následující instrukcí

Instrukce zastavení:

`HALT` – výpočet je ukončen („regulérně“ zastaven)

Stroj RAM

Příklad programu pro stroj RAM, který pro posloupnost čísel na vstupu spočítá aritmetický průměr (zaokrouhlený dolů) a na výstup vypíše odchylky jednotlivých čísel ze vstupu od tohoto průměru:

```
 1   READ
 2   JZERO 10
 3   STORE *3
 4   ADD 2
 5   STORE 2
 6   LOAD 1
 7   ADD =1
 8   STORE 1
 9   JUMP 1
10   LOAD 2
11   DIV 1
12   STORE 2
13   LOAD =0
14   STORE 1
15   LOAD *3
16   JZERO 23
17   SUB 2
18   WRITE
19   LOAD 1
20   ADD =1
21   STORE 1
22   JUMP 15
23   HALT
```

Stroj RAM a Turingův stroj

Každý program v každém programovacím jazyce by mohl být realizován jako program stroje RAM.

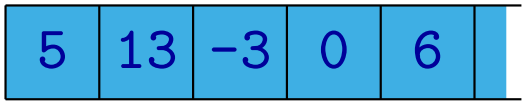
Není složité (i když je to trochu pracné) si rozmyslet, že libovolný algoritmus prováděný strojem RAM je možné realizovat také Turingovým strojem.

Turingův stroj je schopen realizovat libovolný algoritmus, který by bylo možné zapsat jako program v nějakém programovacím jazyce.

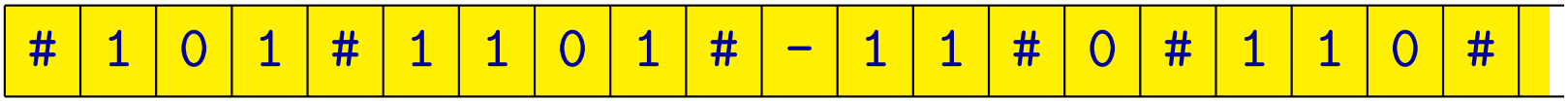
Stroj RAM a Turingův stroj

Turingův stroj pracuje se slovy nad nějakou abecedou, zatímco stroj RAM s čísly. Čísla ale můžeme zapisovat jako sekvence symbolů a naopak symboly nějaké abecedy můžeme zapisovat jako čísla.

Například následující vstup stroje RAM

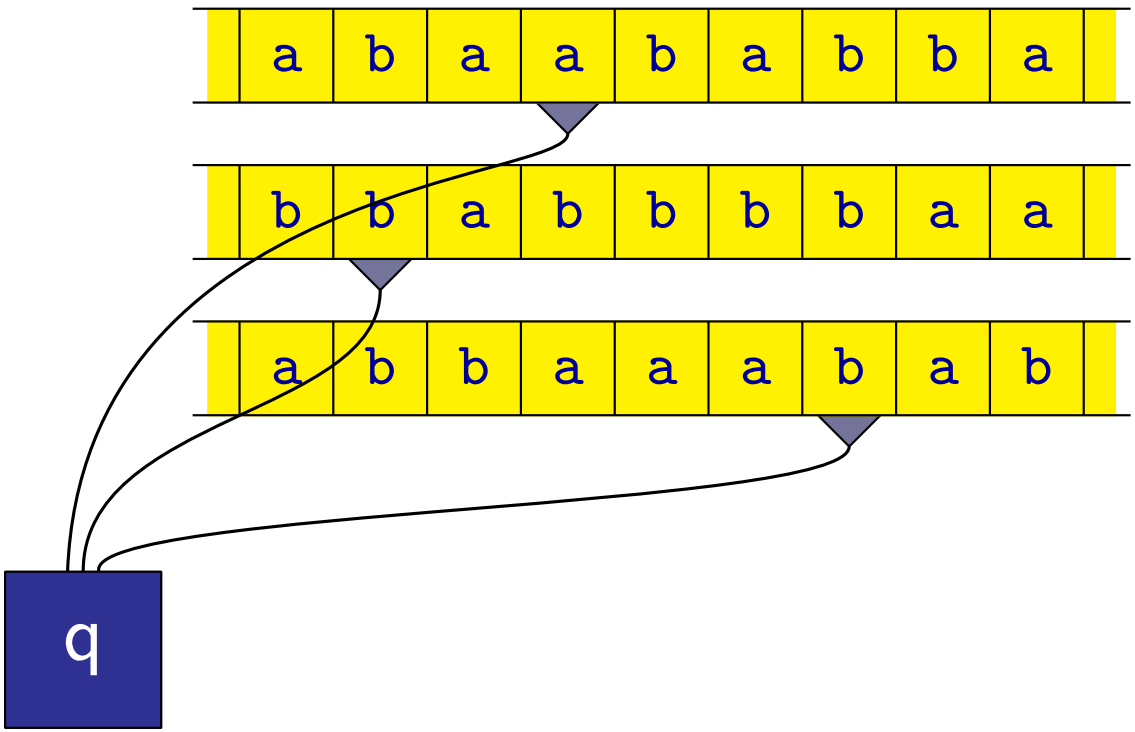


může být v případě Turingova stroje reprezentován jako



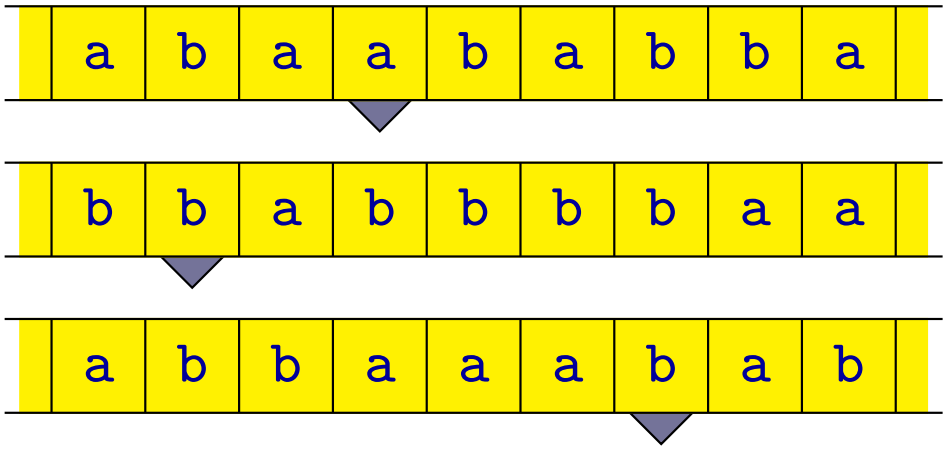
Vícepáskový Turingův stroj

Při konstrukci Turingova stroje k danému stroji RAM může být jednodušší zkonstruovat nejprve **vícepáskový Turingův stroj**:

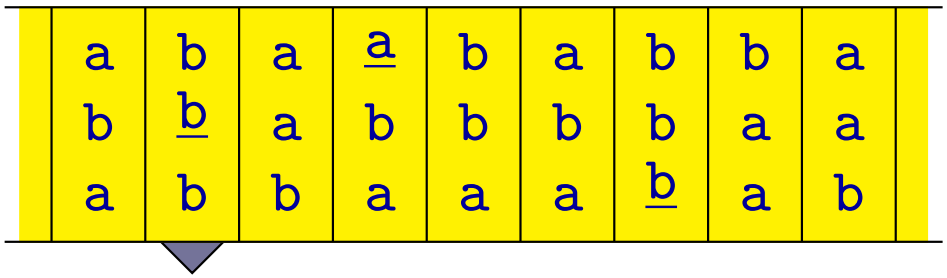


Vícepáskový Turingův stroj

Činnost vícepáskového Turingova stroje

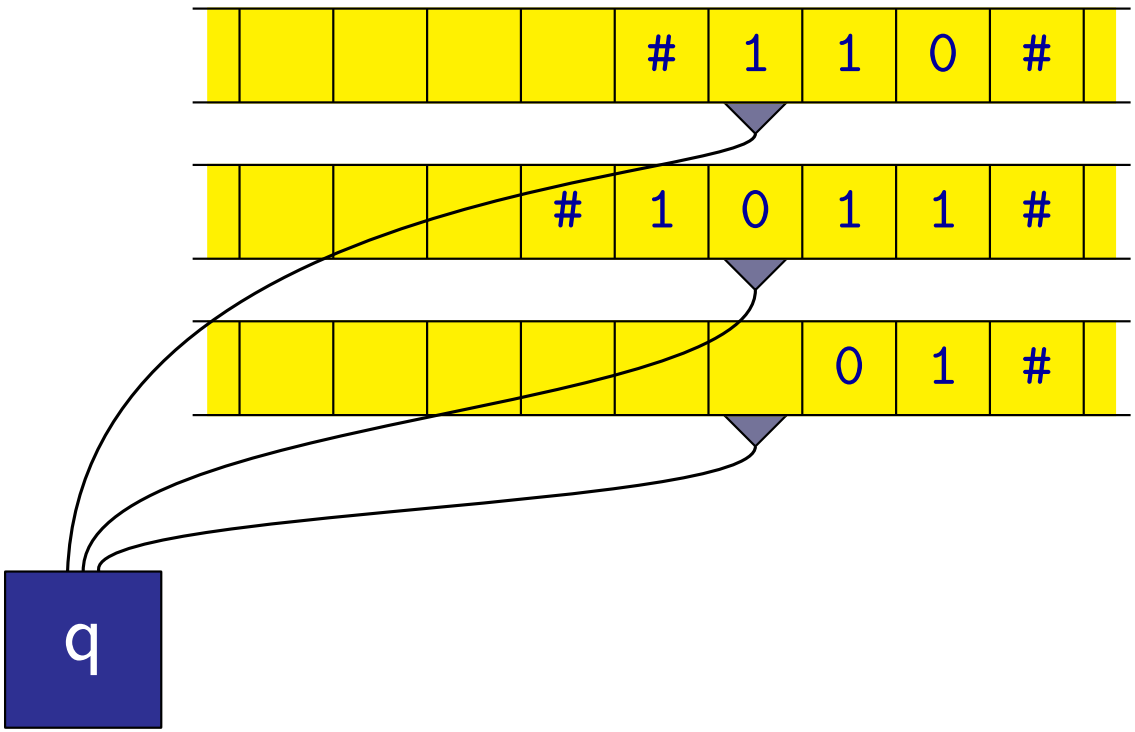


je možné simulovat jednopáskovým Turingovým strojem



Vícepáskový Turingův stroj

Příklad: Stroj, který dostane jako vstup dvě přirozená čísla zapsaná binárně a oddělená znaky # (např. čísla 6 a 11 budou zapsaná jako slovo "#110#1011#") a spočítá jejich součet.



Turingův stroj simulující činnost stroje RAM

Turingův stroj simulující činnost stroje RAM bude mít několik pásek:

- Pásku reprezentující vstupní pásku stroje RAM.
- Pásku reprezentující výstupní pásku stroje RAM.
- Pásku, na které bude uložen obsah pracovního registru.
- Pásku, na které bude uložen obsah indexového registru.
- Pásku, na které bude uložen obsah ostatních buněk paměti stroje RAM.
- Pásku, na které bude uložena adresa buňky paměti, se kterou se aktuálně pracuje.
- Několik dalších pomocných pásek (pro uložení mezivýsledků operací apod.)

Turingův stroj simulující činnost stroje RAM

Turingův stroj si bude v řídící jednotce pamatovat, která instrukce stroje RAM se právě provádí.

Provedení většiny instrukcí není složité:

- Instrukce `READ` – zkopírování hodnoty (ohraničené znaky "`#`") ze vstupní pásky na pásku reprezentující pracovní registr.
- Instrukce `WRITE` – zkopírování hodnoty pracovní registru na výstupní pásku.
- Instrukce `JUMP` – změní se jen stav řídící jednotky Turingova stroje.
- Instrukce `JZERO` a `JGTZ` – snadno otestujeme obsah pracovního registru a podle výsledku změníme stav řídící jednotky Turingova stroje.

Turingův stroj simulující činnost stroje RAM

Aritmetické instrukce (`ADD`, `SUB`, `MUL`, `DIV`) jsou také relativně jednoduché:

- Hodnotu druhého operandu zapíšeme na pomocnou pásku.
- Provedeme operaci (např. sčítání) bit po bitu, výsledek ukládáme na další pomocnou pásku.
- Výsledek zkopírujeme na pásku s obsahem pracovního registru.

Poznámka: Násobení je možné realizovat pomocí série sčítání a bitových posunů.

Turingův stroj simulující činnost stroje RAM

Asi nejsložitější je realizace paměti stroje RAM.

Jednou z možností je pamatovat si jen obsah těch buněk, se kterými stroj RAM v průběhu své činnosti někdy pracoval (víme, že všechny ostatní obsahují hodnotu 0).

Příklad: Stroj RAM zatím pracoval jen s buňkami 2, 3 a 6:

- Buňka 2 obsahuje hodnotu 11.
- Buňka 3 obsahuje hodnotu -1.
- Buňka 6 obsahuje hodnotu 2.

Obsah pásky Turingova stroje reprezentující buňky paměti stroje RAM bude následující:

$	#	1	0	:	1	0	1	1	#	1	1	:	-	1	#	1	1	0	:	1	0	#	$	

Turingův stroj simulující činnost stroje RAM

Instrukce `LOAD`:

- Hledanou adresu uložíme na příslušnou pásku obsahující adresu, se kterou se aktuálně pracuje.
- Budeme hledat příslušnou adresu na pásce reprezentující obsah paměti stroje RAM.
  (Pokud ji nenajdeme, přidáme ji na konec, s tím, že obsahuje hodnotu 0.)
- Příslušnou hodnotu zkopírujeme na pásku reprezentující pracovní registr.

Turingův stroj simulující činnost stroje RAM

Instrukce `STORE`:

- Podobně jako u instrukce `LOAD` najdeme příslušné místo na pásce.
- Zbytek pásky s obsahem paměti stroje RAM zkopírujeme na pomocnou pásku.
- Na příslušné místo zkopírujeme hodnotu z pracovního registru.
- Zbytek pásky, který jsme zkopírovali na pomocnou pásku, zkopírujeme zpět (za nově zapsanou hodnotu).

Churchova-Turingova teze

Churchova-Turingova teze

Každý algoritmus je možné realizovat nějakým Turingovým strojem.

Není to věta, kterou by bylo možno dokázat v matematickém smyslu – není formálně definováno, co je to algoritmus.

Tezi formulovali nezávisle na sobě v polovině 30. let 20. století Alan Turing a Alonzo Church.

Churchova-Turingova teze

Ve stejné době bylo navrženo několik různých formalismů zachycujících pojem algoritmus:

- Turingovy stroje (Alan Turing)
- lambda kalkulus (Alonzo Church)
- rekurzivní funkce (Stephan Kleene)
- produkční systémy (Emil Post)
- ...

Dále můžeme uvést:

- Libovolný (obecný) programovací jazyk (jako např. C, Java, Lisp, Haskell, Prolog apod.).

Všechny tyto modely jsou ekvivalentní z hlediska algoritmů, které jsou schopny realizovat.

Minského stroj

Jedním z nejjednodušších strojů, které jsou schopny realizovat libovolný algoritmus, je tzv. **Minského stroj** neboli **stroj s čítači**:

- Má nějaký pevně daný počet čítačů $x_1, x_2, \ldots, x_k$.
- Každý čítač x_i obsahuje (libovolně velké) přirozené číslo.
- Program je sekvence instrukcí z nichž poslední je instrukce `HALT`:

$$\begin{aligned} 1: &\quad \texttt{instr}_1 \\ 2: &\quad \texttt{instr}_2 \\ &\quad \vdots \\ n-1: &\quad \texttt{instr}_{n-1} \\ n: &\quad \texttt{HALT} \end{aligned}$$

Minského stroj

Kromě instrukce `HALT` jsou všechny ostatní instrukce jen následujících dvou typů:

- $x_i := x_i + 1;$ `GOTO` j

  Například:

  $7:\ x_2 := x_2 + 1;$ `GOTO` 12

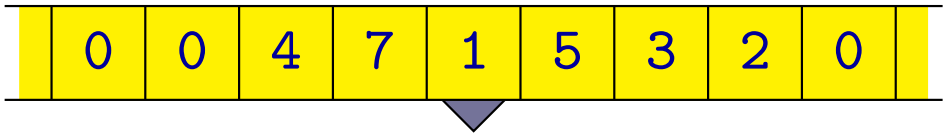
- `IF` $x_i = 0$ `THEN GOTO` j `ELSE` $\{x_i := x_i - 1;$ `GOTO` $k\}$

  Například:

  $8:$ `IF` $x_3 = 0$ `THEN GOTO` 9 `ELSE` $\{x_3 := x_3 - 1;$ `GOTO` $17\}$

Minského stroj

Minského stroj může simulovat činnost Turingova stroje:



Předpokládejme, že pásková abeceda Turingova stroje je $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ a že 0 reprezentuje prázdný symbol „blank“.

Obsah pásky můžeme reprezentovat třemi čísly (píšeme je desítkově):

- Obsah pásky nalevo od hlavy: 47
- Obsah pásky napravo od hlavy: 235
- Obsah políčka v místě, kde se nachází hlava: 1

Minského stroj

Minského stroj může snadno realizovat následující operace:

- Vynulování čítače.
- Přesunutí obsahu jednoho čítače do druhého (s vynulováním původního čítače).
- Vynásobení obsahu čítače konstantou.
- Vydělení obsahu čítače konstantou a zjištění zbytku po tomto dělení.

Pro simulaci Turingova stroje tedy postačuje Minského stroj se třemi čítači (dva čítače pro reprezentaci obsahu pásky a jeden pomocný čítač).

Minského stroj

Činnost Minského stroje s více čítači (např. 3 čítače označené x, y, z) je možné simulovat Minského strojem se **dvěma** čítači:

Hodnoty čítačů x, y, z zakódujeme do jediného čítače jako hodnotu

$$2^x 3^y 5^z$$

Například hodnoty $x = 3$, $y = 2$, $z = 2$ jsou reprezentovány jako hodnota

$$2^3 3^2 5^2 = 8 \cdot 9 \cdot 25 = 1800$$

(Zvýšení x o 1 je pak realizováno jako násobení dvěma, zvýšení y o 1 jako násobení třema apod.)

Druhý čítač používáme jako pomocný čítač pro realizaci těchto operací.